Real-time control of a large swarm of mini-robots for image display

Author(s):
Haag, Stefan

Publication Date:
2010

Permanent Link:
https://doi.org/10.3929/ethz-a-010075315

Rights / License:
In Copyright - Non-Commercial Use Permitted
Real-time control of a large swarm of mini-robots for image display

Autumn Term 2010
# Contents

Abstract ix  
1 Introduction 1  
1.1 Motivation 1  
1.2 Problem statement 1  
1.3 Previous work 2  
1.4 Outline of the thesis 3  
2 Real-time control 5  
2.1 Goal Assignment 6  
2.1.1 Kuhn-Munkres assignment algorithm 6  
2.1.2 Auction Algorithm 6  
2.1.3 Optimized Auction Algorithm 7  
2.1.4 Comparison 8  
2.1.5 Parallelization 8  
2.2 Multithreading 9  
2.2.1 OpenMP 9  
2.2.2 Application of multithreading 9  
3 Characterization 13  
3.1 Introduction to the robot 13  
3.1.1 Properties and parameters of the robots 13  
3.1.2 Battery 14  
3.2 Characterization Process 17  
3.3 Collision avoidance 20  
4 Tracking 23  
4.1 Robot detection 24  
4.1.1 Image acquisition 24  
4.1.2 IR detection 27  
4.1.3 Robot extraction 28  
4.1.4 Robot assignment 30  
4.2 Tracking of the robots 31  
4.2.1 Initialization 31  
4.2.2 Tracking process 32  
4.3 Recovery 33  
5 Autonomous Recharging 35  
5.1 Charging station 35  
5.2 Recharging algorithm 37
6 Results

6.1 Horizontal ................................................. 41
   6.1.1 Trajectory on a square ......................... 41
   6.1.2 Collision avoidance test ..................... 43
   6.1.3 Static pattern formation ..................... 45
   6.1.4 Dynamic pattern formation ................... 50
   6.1.5 Video display ..................................... 51
   6.1.6 Charging ........................................... 54

6.2 Vertical ................................................... 55
   6.2.1 Trajectory on a square ......................... 55
   6.2.2 Collision avoidance test ..................... 56
   6.2.3 Static pattern formation ..................... 57
   6.2.4 Dynamic pattern formation ................... 58
   6.2.5 Video display ..................................... 60
   6.2.6 Charging ........................................... 61

7 Conclusion and Outlook .......................... 63
   7.1 Conclusion .......................................... 63
   7.2 Outlook ............................................. 64

A Datasheet robots ........................................... 68
# List of Figures

1.1 Display Swarm with 75 robots displaying three stars .................. 2

2.1 Computation time for goal assignment ................................. 8
2.2 Multithreading Flowchart ............................................. 12

3.1 Minipuck robot ......................................................... 13
3.2 Battery level of robots during run (unfiltered) ......................... 15
3.3 Battery level of robots during run (filtered) .......................... 16
3.4 Battery level of robots during waiting ................................ 16
3.5 Battery level of robots during charging ............................... 17
3.6 Power control: Velocity and angular velocity .......................... 18
3.7 Velocity control: Velocity and angular velocity ....................... 19
3.8 Optimized Velocity control: Velocity and angular velocity .......... 19
3.9 $S_{AHV}$ for fixed $\varepsilon$ and varying $T$ ............................ 21

4.1 Process of robot detection for tracking ................................. 23
4.2 Experimental setup .................................................... 25
4.3 Captured image by the camera ........................................... 26
4.4 Shifted IR LED because of the perspective ............................. 28
4.5 Two robots with one IR LED on ....................................... 29
4.6 Illuminant searching process ........................................... 30
4.7 Pose detection of two robots .......................................... 31
4.8 Matching in the tracking process ......................................... 33
4.9 Robot recovery ........................................................... 33

5.1 A charging station with three charging slots .......................... 36
5.2 A charging slot and the bottom of a robot ............................. 36
5.3 Additional guidance for the charging stations ........................... 36
5.4 Steps showing a robot approaching a charging station ............... 38

6.1 Trajectories on horizontal surface ...................................... 42
6.2 Collision avoidance ...................................................... 44
6.3 Formation of a circle with 75 robots .................................. 45
6.4 Goal assignment with manual movement of one robot ................ 46
6.5 Image of a worm in an apple .......................................... 47
6.6 Robots displaying different simple patterns .......................... 48
6.7 Trajectories between static patterns .................................... 48
6.8 Robots displaying different patterns from images .................... 49
6.9 Dynamic pattern formation: Circle with 75 robots .................... 50
6.10 First frame of each part of the video ................................ 51
6.11 Video display with 50 agents (1/2) ................................... 52
6.12 Video display with 50 agents (2/2) ................................... 53
6.13 Statistics about the charging approach ................................ 54
6.14 Trajectories on vertical surface ........................................ 56
6.15 Static pattern formation on the vertical surface .................... 57
6.16 Trajectories between static patterns on the vertical plane .......... 58
6.17 Dynamic pattern formation on the vertical surface ................. 59
6.18 Vertical video display with 14 agents ............................... 60
List of Tables

2.1 Computation time for goal assignment ................. 8

3.1 Parameters of the robot .................................. 14
3.2 Power parameters of the robot .......................... 14

6.1 Parameters used for horizontal usage .................... 42
6.2 Parameters used for vertical usage ....................... 55
6.3 Deviation of the robots to the direct straight line 56
Preface

This master thesis was realized at the Autonomous System Lab (ASL) at ETH Zürich in cooperation with Disney Research Zurich (DRZ). I would like to thank Javier Alonso-Mora and Andreas Breitenmoser for their support and assistance during the whole project. Further, a thank to Dr. Paul Beardsley from DRZ for his support, Dr. Gilles Caprari for the production of the minipuck robots with charging stations and his support throughout the project and all the other people form ASL that supported me. I want also to thank Prof. Roland Y. Siegwart who made this interesting project possible.
Abstract

This thesis presents an advancement and extension of the previous work in multi-agent robotics. This novel and innovative technology can be used to display information, images, animations or videos where each robot represents one pixel. A large swarm of these robots is not just able to display almost any desired image, it can also display choreographic motion, dynamic image representation and dynamic video display.

In this thesis, real-time control of a swarm of 75 small autonomous mobile robots is achieved. For that, existing algorithms of the previous version have been adapted, extended and optimized for using them to control a large swarm of robots. Further, new robots are used that are much cheaper and less accurate than the previous ones. Due to the indistinguishability of the new robots, a tracking algorithm is implemented. The handling of the new robots is simplified by implementing charging stations allowing the robots to charge autonomously. Further, the method is ported to a vertical surface where the robots drive on a wall which allows better visibility of the displayed pattern.

The approach presented here is developed for the entertainment industry but the working principle of the multi-agent system can be used for many different applications ranging from collaborative exploration and coverage to optimal task allocation of resources.
Chapter 1

Introduction

This thesis presents an advancement and extension of a previous work in multi-agent robotics [1] facing new challenges. The approach presented here is developed for entertainment, and in particular for image formation, but the working principle of the multi-robot system can be used for many different applications.

1.1 Motivation

Today, for displaying information or animations, large 2D screens are very common. They allow the simple display of any information or video. Due to their wide presence they do not have the potential to be an eyecatcher anymore. Further, they just allow displaying information on a fixed 2D grid. A novel and innovative technology to display information, images, animations or videos is presented in this work: Display Swarm. It consists of a large number of small autonomous robots where each robot represents one pixel. A large swarm of these robots is not just able to display almost any desired image, it can also display choreographic motion, dynamic image representation and dynamic video display which is not possible with a common 2D screen. This novelty is not known yet and has the potential to be apprehended by the guests in Theme Park or also in other situations. Known from videos of large swarms of birds or fishes, most of the people are fascinated from swarms moving in a choreographic motion. In a further step, it is possible to add an interaction of the human with the swarm. Possibilities are that someone could draw a figure which is then displayed by the robot swarm or that the robots directly follow the movements of the person.

In the future, the technology of swarm robots could also be used to solve many different problems for which execution, many small robots working together in a swarm could be beneficial.

The result of this work can be seen in Figure 1.1 where 75 robots display a pattern showing three stars.

1.2 Problem statement

The goal of the project is to develop a novel robot display and to achieve real-time control of a swarm of 75 small autonomous mobile robots. Together, they are able to display images and videos where each robot represents a pixel of the image. Starting with a working version with fourteen e-puck robots [1] the algorithms have to
be adapted, extended and optimized to achieve real-time control of the larger swarm.

The main work packages of this project are:

- Port the existing system to a high-performance quadcore computer
- Evaluation of how the system scales to larger number of robots
- Characterization of the new robots
- Detection of the new robots
- Tracking and controlling the robots in real-time
- Implementation of a distributed goal assignment
- Extend algorithms to multi-thread computation
- Adapt the existing collision avoidance algorithm
- Implement autonomous recharging of the robots in charging stations
- Adapt the Display Swarm for vertical usage
- Adaptation and optimization of the User Interface

The main challenge is to be able to handle and control a large swarm of mini-robots that are built much cheaper and simpler than the previous version and are therefore less accurate. Further, they are faster and not uniquely distinguishable.

1.3 Previous work

This master thesis mainly builds on the master thesis of Javier Alonso-Mora [1] and continuing work [2] and [3]. The previous version used a maximum of fourteen e-puck robots that were controlled in real-time and were able to display different
patterns collision free in a visually appealing way. Extension to larger groups was presented solely in simulation. The number of robots was limited by the high cost of the robots, the computational expensive algorithms, limited communication speed and the computation power of the computer, where just a single core was used. Further, they were too large and expensive to use them in a large swarm.

1.4 Outline of the thesis

This work is structured as follows:

Chapter 2 presents the different methods for the goal assignment and the explanation for the multithreading that is used to improve the speed of the algorithms. Chapter 3 introduces the robots used for the demo and presents the characterization process as well as the adjustments that are made in the collision avoidance algorithm. Chapter 4 presents the different steps needed to track the robots, starting with the image acquisition, IR detection, robot extraction and robot assignment and continuing with the tracking algorithm. Chapter 5 presents the autonomous charging that is used to automatically charge the robots during the demos. Chapter 6 shows the most important results of the thesis. Chapter 7 concludes and shows the future outlook.
Chapter 2

Real-time control

To achieve real-time control for a large number of robots, fast algorithms are needed. With the 14 robots of the previous version, real-time control at 10 Hz was possible but additional robots increase the computation time of the algorithms. The algorithm can be divided in four independent parts, all of them presenting a non-negligable computational cost, as follows:

1. Image acquisition (incl. robot detection)
2. Sending of the commands via radio
3. Goal assignment
4. Collision avoidance (NH-ORCA)

Therefore, the focus is to improve these parts to achieve real-time control of a large swarm of robots, in particular of at least 75 robots at 10 Hz in a Intel quadcore computer.

Further analyses showed that the computation time for image acquisition and robot detection does not increases much by adding more robots. For a single robot, this process takes about 0.015 seconds where for 50 robots, it lasts about 0.025 seconds. But because it still needs a lot of computation time and blocks therefore the other algorithms during that time, it is extracted from the main algorithm and is computed in a separate thread. For further information about multithreading see Section 2.2 and Figure 2.2 where a scheme of the whole algorithm is presented.

The sending of the commands via radio uses by far the most time in the version with fourteen robots. The time just increases linearly by adding more robots but to use it with a large swarm of robots, mainly two changes are done. First, this process is extracted from the main algorithm and computed in a separate thread. Furthermore, several packages are sent at once now. For further information see Section 2.2.

To improve the collision avoidance, the algorithm was modified to use in multi-threading (see Section 2.2).

To increase the speed of the goal assignment algorithm, a completely different algorithm is presented.

The generation of the goal positions is another important part of the system but because this is done offline in previous, it is not a part of the real-time control.
Chapter 2. Real-time control

2.1 Goal Assignment

To solve this optimization problem, two different algorithms are implemented, the ‘Kuhn-Munkres assignment algorithm’, also known as the ‘Hungarian algorithm’ [4] [5] (which is already implemented in the previous version [1]) and the Auction algorithm [6]. The aim of the assignment is that every robot is uniquely assigned to a goal position in a way that the total cost is minimized. The cost of each robot being assigned to each goal position is defined as the squared distance to its goal. Thus, the objective is to minimize the sum of squared distances to the goal. To determine the assignment, both algorithms need a cost matrix where all the computed squared distances between every robot and every goal are stored. The algorithms then minimize the total cost.

2.1.1 Kuhn-Munkres assignment algorithm

The Munkres assignment algorithm finds always the optimal solution and therefore the assignment where the costs are minimal. Further information of this optimization method can be found in the works by Kuhn [4] and Munkres [5]. More details about the implementation in this project can be found in [1]. The large disadvantage of this algorithm is its computation cost of at best $O(n^3)$. Where this is negligible for a low number of robots, the algorithm gets very computational expensive for a larger number of robots. For too many robots, the real time computation of the assignment is not possible anymore. In a first step, the runtime can be reduced by using the computed optimal assignment of the previous time step as initial guess for the new assignment. Another possibility is that the assignment is not computed in every step but this can cause instabilities of the system. Because of its centralized architecture, further optimization by parallelization is not possible.

2.1.2 Auction Algorithm

In contrast to the Kuhn-Munkres assignment algorithm, the Auction algorithm iteratively solves the assignment problem by finding a sub-optimal solution, where the error committed is specified as a design parameter. Trade-off exists between optimality of the solution and computation time. Furthermore, the number of iterations is finite and an upper bound can be found [7]. This method has been seen to be considerably faster than the Hungarian Algorithm. For further information, see [6].

The Auction algorithm works similar to a market with bidders and objects they bid for. The prices of the objects are changed during the auction. The bidders are represented by the robots where the goals are the objects they bid for. The auction finishes if every robot has auctioned exactly one goal. To achieve that, several auction rounds are performed where all unassigned robots bid for the goal with lowest cost. After every round, the goals that have received a bid are allocated to the robot that offered the highest price for it.

The algorithm contains mainly two phases (for details refer to[6]):

Bidding phase

In the bidding phase, every robot $i$ finds the goal $j$ from a given subset $A(i)$ of goals where its difference between the benefit $a_{ij}$ and the price $p_j$ is maximized.

$$j = \arg\max_{j \in A(i)} \{a_{ij} - p_j\}$$  \hspace{1cm} (2.1)
The benefit $a_{ij}$ is in this case the negative squared distance between the robot and the goal so that the benefit is high for robots that are close to the goal. The bid itself consists of the price $p_j$ and a positive bidding increment $\gamma_j$

$$bid = p_j + \gamma_i$$  \hspace{1cm} (2.2)

For every bidding increment $\gamma_j > 0$, the iteration process terminates after a finite number of iterations. For small $\gamma_j$, the auction algorithm gets more optimal but it also gets slower than for larger $\gamma_j$. To determine $\gamma_j$, a positive increment $\delta$ is chosen so that

$$\gamma_i = \delta + v_i - w_i$$  \hspace{1cm} (2.3)

with the best value for the goal $j_i$

$$v_i = \max_{j \in A(i)} \{a_{ij} - p_j\}$$  \hspace{1cm} (2.4)

and the second best value for the goal $j_i$

$$w_i = \max_{j \neq j_i, j \in A(i)} \{a_{ij} - p_j\}$$  \hspace{1cm} (2.5)

Further, the number of iterations to the end of the assignment tends to be proportional to $C/\delta$ [7], where

$$C = \max_{(i,j) \in A} |a_{ij}|$$  \hspace{1cm} (2.6)

Optimization is possible by using $\delta$-scaling, that means applying the algorithm multiple times with decreasing $\delta$. Depending on the initial value of $\delta$ and the decreasing factor, the performance of the algorithm can be adjusted. A large $\delta$ results in a higher computation time but optimal solution where a small $\delta$ decreases the computation time and makes the solution suboptimal.

**Assignment phase**

All the goals that receive bids are assigned to the robot that placed the highest bid. In a further step, its price $p_j$ is increased to the highest bid. If there is already a robot assigned to that goal from a former bidding round, it gets unassigned and can take part in the next bidding round. The price and the assignment of goals that does not receive any bid in the current round remain unchanged for the next bidding round.

### 2.1.3 Optimized Auction Algorithm

The Auction Algorithm can be further optimized. The idea behind this optimization is that the assignment usually does not change much during one time step. Therefore, the prices used in the Auction Algorithm are saved in every time step and used as initial values for the next time step. Further, $\delta$ is set to a very low value which reduces the number of iterations dramatically. This can increase the speed of the algorithm as long as there are not too many changes. If there are large changes, for example if the pattern and therefore all goal positions change, the algorithm takes the values for the prices and the $\delta$ that were used before the optimization.

With this optimization, the risk that the algorithm needs much too long to find a solution is increased. This happens if the assignment and the prices from the last step are very bad in the current time step. To avoid that the algorithm takes too much time and delays the rest of the code, the maximum number of iterations is limited. If this maximum is reached, the algorithm takes the old assignment for the current time step. For the next one, the goals are handled as complete new ones with initial prices and a larger $\delta$. 


2.1.4 Comparison

The comparison of the Munkres assignment algorithm and the Auction algorithm shows that the average computation time for a large number of robots is much smaller with the Auction algorithm (Table 2.1). To achieve visually appealing trajectories, the $\delta$ is chosen smaller than proposed in [8]. That decreases the computation speed which can be seen especially for small number of robots where the Munkres assignment algorithm becomes faster than the Auction algorithm. Nevertheless, the computation time is still very low for large number of robots. Figure 2.1 shows the computation time needed for each iteration for a Matlab simulation with 1000 robots. The lower computation time of the optimized auction algorithm compared with the auction algorithm in the first step is due to a very low $\delta$ for the auction algorithm and a higher $\delta$ in the first step for the optimized auction algorithm.

2.1.5 Parallelization

There are possibilities to parallelize the auction algorithm (see [8]). Due to the fact that everything is computed centralized on one single computer, the parallel code
would have to communicate a lot, which decreases the speed significantly. Further, for 75 robots the optimized auction algorithm is already very fast. Therefore, the goal assignment is not parallelized.

2.2 Multithreading

The used algorithms are computational expensive. Due to the fact that the computation in this case is all done centralized for all robots, significant computation power is needed. In the previous version only one core was used. The other cores of multicore computers were not used. To use the full amount of resources, the code is split up in threads that can be executed in parallel. This allows running the algorithms fast and therefore to achieve real-time control at 10 Hz with 75 robots. This can be done for example with OpenMP.

2.2.1 OpenMP

The tool that is used to extend the algorithms to multithreading is OpenMP \(^1\) (Open Multi-Processing) which supports multiprocessing programming in C++. OpenMP allocates threads to the different cores of the processor. That means that one core can execute more than one thread at the time, depending on the computation power that is needed for each thread. The allocation is done automatically and does not have to be considered when programming. The different threads have to be previously defined. They can split the code in different sections running in parallel or divide a for-loop in parts that are processed in parallel. Further information about multithreading with OpenMP can be found in [9] and [10].

The main advantages of OpenMP are that it is relatively simple to implement and that the general structure of the code does not have to be significantly changed. The disadvantage is that the parallelization is done on a very high level and therefore some processes (for example which thread is allocated to which core) cannot be controlled. Further, multithreading makes debugging much harder and there is the risk of creating race conditions.

One of the main issues using OpenMP is the handling with shared data. Generally, OpenMP allows that several threads read and write to the same memory at the same time. This causes conflicts and can result to wrong results. Further, it slows down the program. This means that data sharing between the threads should be minimized. In this project, it never happens that two threads write in the same memory but it happens that one thread writes and another reads the memory. It shows that this is not a problem.

2.2.2 Application of multithreading

The multithreading scheme can be seen in Figure 2.2. First, an initialization is done and the data is loaded. This is done in a single thread because it is not time-critical and not computational intensive. The multithreading starts as soon as the main algorithm starts, splitting up the code in three main threads.

\(^1\)http://openmp.org/
Sending commands to the robots

The sending thread just uses the velocity, angular velocity and color values that have been computed from the main thread, determines the velocity of each wheel and converts the data into signals that are sent to each robot. The specialty of the sending thread is that the performance is mainly limited by the USB communication with the radio where the computation takes just very short time. By executing it in a separate thread, the computation power can be used by other algorithms and the main code is not slowed down by waiting for the USB communication. In a further optimization, the signals for four robots are sent to the radio at once which improves the speed because the limit of the USB communication is not the data throughput (which is very low in this case) but the quick changes between sending and receiving commands (which is limited by the USB protocol).

The sending thread is synchronized with the main thread. After the signals are sent the each robot, the thread waits until there is new data from the main thread. Therefore, because the main thread is running at 10 Hz, the sending thread also runs at 10 Hz.

Robot detection

The thread for grabbing the image takes the image from the camera as input and outputs the position an orientation of the robots on the display area in real coordinates in meters and in pixel coordinates. The different steps to achieve that are:

1. IR detection
2. Robot extraction
3. Compute detected position of all robots
4. Compute position estimation of all robots
5. Fuse detected and estimated position (Tracking)
6. Save position and orientation for each robot

Further details about this process can be found in Chapter 4.

The image grabbing thread is executed at 15 Hz, which is the maximum frequency of the camera. This thread is therefore not synchronized with the main thread. The higher frequency means that the main thread does not use every computed position of this thread. This is done to assure that the main thread always gets the most actual data of the robot poses.

Main thread

The main thread mainly takes the robot poses from the image grabbing thread and computes the velocity and angular velocity of each robot to achieve a certain goal. Further, the main loop also controls the other threads (e.g. stops the other threads if the demo finishes). Besides that, the main loop has the following main tasks:

- **Battery control**: At predefined points of the demo, the thread checks the battery status of every robot and assigns the robots to a charging station if needed. Further details about that can be found in Chapter 5.


- **Load goal position**: Depending on the number of robots running in the demo (without the ones that are in the charging process), precomputed goal positions are loaded from a file.

- **Goal assignment**: The detected robots are assigned to a goal position as described in 2.1.

- **Set colors**: Besides the goal for each robot, the color of its LED has to be set. This can be done with precomputed colors depending on the goal or position of each robot.

- **NH-ORCA**: The NH-ORCA is the local collision avoidance which is described below.

- **Set the velocities**: The linear ($v$) and angular ($\omega$) velocities are set as outputs from the local collision avoidance.

- **Check if finished**: At the end of the loop, it is checked if the simulation already finished. The demo finishes if the final positions are reached, the running time is expired or the user manually stops it.

The NH-ORCA (for further information see [3]) is the most computationally expensive part of the main loop. Therefore, this part is split up in several threads which compute the collision avoidance for each robot in parallel. For each robot, a preferred velocity $\vec{v}_{\text{pref}}$ is computed depending on the position and velocities of the other robots. This can be easily split up because for each robot, the same data is used and there is no direct communication between the different threads. With some other optimizations in the NH-ORCA algorithm, the increase in speed allows the main loop to run at 10 Hz with 75 robots.
Figure 2.2: The program is divided in three main threads: the first grabs and evaluates the image, the second sends the signals to the robots and the third does the main computation needed for the demo. In the third thread, the NH-ORCA collision avoidance is further divided in different threads.
Chapter 3

Characterization

3.1 Introduction to the robot

The mobile robot used for the experiments is called Minipuck and can be seen in Figure 3.1. It is developed for this purpose and replaces the previously used modified e-puck. The Minipuck robots are manufactured by GCtronic\textsuperscript{1}. Further details and properties of the robots and the internal controller can also be found in the datasheet in the Appendix of this thesis.

![Minipuck robot with color LED on. The three blue dots on the perimeter of the circle are the IR LEDs](image)

Figure 3.1: Minipuck robot with color LED on. The three blue dots on the perimeter of the circle are the IR LEDs

3.1.1 Properties and parameters of the robots

As the e-puck, the Minipuck is also a differentially driven robot and it contains a radio communication system. With this communication system it is possible to control up to 200 robots at a frequency of 10 Hz. Instead of the light emitting panel with 9 independent RGB LEDs of the e-puck, a single RGB LED panel which can be lit in any desired color is used for the Minipuck. Additionally, each robot

\textsuperscript{1}http://www.gctronic.com/
is equipped with 3 infrared LEDs for detection and tracking. In comparison with the e-puck which had 8 infrared LEDs, it is not possible to create different patterns with them to detect each robot uniquely. Further, the Minipuck is smaller and faster than the e-puck and it contains no sensors except an accelerometer for vertical usage. The robot is moved by two wheels powered by two independent motors, achieving a maximum speed of 0.6 m/s and a maximum rotation velocity of about 30 rad/s. For practical reasons, the velocity and the angular velocity are limited to lower values for the Display Swarm. Several constants of the robot are presented in Table 3.1.

Another feature of the Minipuck is that it has magnetic wheels. These allow the robot to drive on a vertical surface which can be interesting for demos of the Display Swarm. To compensate the problem with the additional gravitation, a controller in the robot uses an accelerometer to detect the direction of the robot and adjusts the power of the motors so that the robot can still move in every direction at a desired speed.

### 3.1.2 Battery

For the energy supply, each robot contains two batteries with a total capacity of 260 mAh. Further details about the power consumption can be found in Table 3.2. Figure 3.2 shows the battery level for different robots running in a long-term test at different velocities. The experiment was done by running five robots for each velocity open-loop in a circle at the given velocity. The battery values of each robot is saved every second. For clarity, the noisy values of Figure 3.2 are filtered and normalized for each robot. The normalization is done by transforming the battery level to a value between 0 and 100. This is shown in Figure 3.3. It can be seen that the effect of velocity change in energy consumption is negligible. Further, it can be seen that the decrease is not linear. For below 40%, the energy level decreases dramatically. This is important to know for planning the recharging so that the robot has always enough energy to reach the charging station.
3.1. Introduction to the robot

Figure 3.2: Decreasing battery level of several robots in at different velocities. The values are unfiltered.

Figure 3.4 shows the unfiltered but normalized battery level of nonmoving robots, with and without turned on the color LED. During the whole measurement time, the robots send and receive signals from the radio, which is the case for demos of the Display Swarm. Standby without communication would need much less energy.

In Figure 3.5 the increasing battery level during the charging process can be seen. As for the running robots, the battery value was saved every second. The values are normalized but not filtered. It can be seen that the charging process is very fast at the beginning and gets slower if the value is higher than about 50%.

In general, the charging is about double as fast as the discharging during the run which allows to have less charging stations and more robots running at the same time. Approximately 33% charging stations are needed for continuous performance of 66% of the robots.

An additional challenge that comes from the large number of robots is their handling. To turn them on and off manually every time would take very long. Therefore, the robots have a sleep mode where they use very few energy ( < 2 mA ) so that they can be left running for a long time. To charge all the robots manually would also take a very long time, so they are equipped with contacts that allow them to charge themselves at charging stations.
Figure 3.3: Decreasing battery level of several robots in at different velocities. The bold line indicates the median value of the single measurements (thin lines). The values are filtered and normalized.

Figure 3.4: Decreasing battery level of several nonmoving robots. The bold line indicates the median value of the single measurements (thin lines). The values are not filtered but normalized.
3.2 Characterization Process

In comparison with the e-puck robots used in the previous version, the new robots are not characterized before. Where the e-pucks can use internal wheel encoders, the new robots haven’t integrated them. To characterize a robot, a constant signal is sent to it and the movement is recorded. From the recorded video, an algorithm is used to extract the IR LED in every frame. With this information, the average velocity over a certain time is computed. Additionally, another algorithm fits a circle in the detected trajectory to determine the angular velocity. This is repeated for different signals to get the complete characterization.

The robot contains two different controllers:

**Power control**

In the power control mode, the robot uses the sent signal and converts it to a certain voltage proportional to the signal to drive the wheels. This method is very simple but it has the large disadvantage that due to inaccuracies in the robots, their trajectories are unpredictable. This means that for the same signal, the two wheels of the robot turn at different speeds which results in driving in a circle when it is expected to follow a straight line. It also showed that the difference in speed is also dependent on the driven velocity and further, it changes over time. Nevertheless, after a characterization, the controllability of the characterized robot was improved. This characterization is only valid for one robot and would have to be done for all of them regularly. Therefore, this is not feasible.

The diagram of the velocity of a sample robot at different signals can be seen in Figure 3.6 on the left. On the right, the angular velocity at different signals is shown. For another robot of the set, the result would be different.
Figure 3.6: Power control of one sample robot driving forward and backward, Left: Velocity of the robot at different signals, Right: Angular velocity at different signals

**Velocity control**

The velocity controller uses a PID controller to determine the actual velocity of the robot. For details see Appendix A. For that, the motor is switched on and off at a very high frequency. During off time, the induced tension is measured. This tension is proportional to the speed. This results in a much better predictable behavior than the power control and the differences between the different robots are much smaller and can therefore be neglected. That means that one characterization can be used for all robots. The angular velocity is still not precisely predictable but the errors are much smaller than in the power control and the remaining error in the angular velocity is eliminated in the closed loop control.

The diagram of the velocity of a sample robot at different signals can be seen in Figure 3.7 on the left. On the right, the angular velocity at different signals is shown. Other robots of the set have very similar characteristics.

It shows that especially at very low speeds, the behavior is still not really predictable. This is a problem especially for the autonomous charging where the robot approaches to the charging station at low speed and high precision is needed. Therefore, the velocity controller is further optimized. The drawback of the optimization is that the velocity is not anymore linear dependent on the signal and the resolution is lower for the used velocities (0 – 0.2 m/s). The diagrams of the optimized velocity controller can be seen in Figure 3.8.

It shows that the optimized velocity control yield in better results than the initial velocity control and much better results than the power control and therefore, the optimized velocity control is used.
3.2. Characterization Process

Figure 3.7: Velocity control of a sample robot, Left: Velocity of the robot at different signals, Right: Angular velocity at different signals.

Figure 3.8: Optimized velocity control of a sample robot, Left: Velocity of the robot at different signals, Right: Angular velocity at different signals. The different colors indicate the measurements of different robots. The blue box indicates the range that is used for Display Swarm.
3.3 Collision avoidance

The collision avoidance is mainly inherited from the previous version [3], which is an extension of the one used in [1], and is called 'Non-holonomic optimal reciprocal collision avoidance (NH-ORCA)'. It allows collision free and smooth movements with non-holonomic robots. The main adaption that is made is the extension to multithreading to speed up the computation time. More about that can be found in Section 2.2. Further, some parameters need to be adapted for the new robot.

In a first step, the holonomic trajectory of each robot is computed. This trajectory starts with a circular segment and then gets a straight line to the goal. This trajectory is recomputed and therefore changed in every time step.

The non-holonomic control can be expressed with the linear velocity $v$ and the angular velocity $\omega$. These values are constraint by the robot (see Table 3.1). The feasible set of the controls $(v, \omega)$ is $S_{NHC}$.

The maximum tracking error of the non-holonomic motion of each robot is a given $\varepsilon$. It is guaranteed that the robot stays always inside $\varepsilon$. In this case, several $\varepsilon$ between 0.0025 m and 0.02 m were tested. It showed that a value of $\varepsilon = 0.01$ m showed the best results.

The set of allowed holonomic velocities $S_{AHV}$ is a set of holonomic velocities that can be tracked with a maximum tracking error of $\varepsilon$. To make the trajectories smooth, the time to reach the final orientation can be limited to a maximum value $T$. Figure 3.9 shows $S_{AHV}$ for different $T$. The best results are reached with $T = 0.2$ s. The robot tries to achieve the correct orientation in the fixed time $T$. If that is not possible, it performs a turn in place to the final orientation. Nevertheless, this will never happen with our current values, unless $\varepsilon = 0$.

The collision avoidance then finds the optimal velocity in the set of allowed holonomic velocities $S_{AHV}$, this is the closest to the preferred velocity that is collision free.

It shows that with the maximum speed of about 0.38 m/s, the robots are not really controllable. Reasons for that are the more difficult detection of the robot if the captured image is blurred because of its velocity and more unpredictable behavior of the robot at higher velocities. Further, in a densely crowded display area, very high velocities are not possible. Therefore, the maximum velocity of the robots is limited to $v_{max} = 0.2m/s$. This has also the advantage that $S_{AHV}$ is limited. Without limitation, $S_{AHV}$ is very narrow for most possible values of $T$. This results in preferred velocities $\vec{v}$ that prevents the robot to corner sharply. This can result in robots that circle a goal instead of driving to the goal.

The limitation of $v$ also limits the max rotation velocity $\omega$ to

$$\omega = \frac{\theta}{T} = \frac{\pi}{0.2s} = 15.7 \text{rad/s}$$

For simplicity, $S_{AHV}$ is approximated by a polygon $P_{AHV}$.

For further information about NH-ORCA refer to [3].
Figure 3.9: $S_{AHV}$ for $\varepsilon = 0.01$ m and varying $T$. Limiting the maximal velocity (orange circle) results in a limited velocity $\vec{v}_{lim}$ with a higher holonomic velocity in $y$ direction $V_{H,max,y}$ compared with the $x$ direction $V_{H,max,x}$ what allows smoother trajectories to the goal.
Chapter 4

Tracking

In comparison with the previously used e-pucks, the new Minipucks are indiscernible from each other. As a consequence of that, they have to be tracked to know the position of each robot at every time. This is essential for controlling them in the large swarm. To track the robots, a multi-step process is executed in every time step. This process can be seen in Figure 4.1. The process starts with the image acquisition from the camera. After that, all IR LEDs are detected in the image. This is required to extract the robots from the image. In a next step, the extracted robots are compared with the predicted positions of the robots knowing the velocity and the angular velocity from each robot. This allows assigning the detected positions to the predicted ones. That information is then used for the tracking algorithm.

![Diagram of the tracking process](image.png)

Figure 4.1: General process used for robot detection to get the position of the robots needed for the tracking process
Chapter 4. Tracking

4.1 Robot detection

For the detection of the robots, a camera pointing on the display area is used. Each robot has the color LED on top and the frontal IR LED on. These two LEDs are used to determine the position and the orientation of each robot during the demo.

The two IR LEDs on the back are not used to detect the robot during the normal display. They are just used to find the initial position of each robot (see Section 4.2.1) and to recover lost robots (see Section 4.3).

4.1.1 Image acquisition

To find the robot poses on the horizontal surface, a monochrome image from a camera with a resolution of 1200 x 1600 pixels is used. The higher resolution in comparison with the previous version, where 600 x 800 pixels were used, is needed because of the different optics that allows covering a larger area with the camera. In the current setup (see Figure 4.2), the camera for the horizontal usage is mounted at a height of 2.3 meters and acquires the whole display area which is 2 x 2 meters. The camera is not mounted perfectly overhead of the area and the lens has a distortion at the corners which is compensated with the calibration. The calibration is done with the ‘Camera Calibration Toolbox for Matlab’ [11]. This results in an accuracy of about 2 mm, depending on the position on the display area. This accuracy is similar to the one from the previous version with the lower resolution, a different optic and therefore a smaller display area. The pixels have values between 0 and 255. A sample image is shown in Figure 4.3.

For the vertical usage, a 1.5 m wide and 2.3 m high metallic sheet that is mounted to the test-setup is used as display area (see Figure 4.2). Further, another similar camera with a resolution of 960 x 1280 pixels and a different optic is used. The accuracy is slightly worse than on the horizontal plane.

The settings of the camera are set manually. The value of the exposure and the shutter time is a tradeoff between good detectability of the IR LEDs, the visibility of the robot shape if the color LED has low brightness and the sharpness of the image if the robots move. A low exposure results in a dark image in which, the IR LEDs can be extracted very easily but the shape of the robot is almost invisible. The shutter time has to be small enough that robots in motion are still shown sharp because the IR LED detection fails if the image is blurred. The drawback of a small shutter time is a noisy and dark image which can also result in problems with the detection.

Generally, the settings are also dependent on the ambient light, for example the illumination of the room. It shows that the detection of the IR LEDs is not susceptible to changes in the illumination if the correct camera settings are chosen. The same applies to the extraction of the robot position if the color LED in on. If this is off, a low illumination makes it very hard to find the robot.
Figure 4.2: Experimental setup
Figure 4.3: Sample of a captured image by the camera with a small shutter time and a low exposure for robot detection showing 75 robots forming a circle.
4.1.2 IR detection

In a first step, all IR LEDs are searched over the whole image. A very similar function was already implemented in the previous version [1] and is therefore also used in the actual version with some minor changes. The detection algorithm works as follows:

1. Find all pixels of the image with a value over a certain threshold. Because of the brightness of the color LED can be higher than the one of the IR LED, both, color LED and IR LED pixels are found in this step.

2. Pseudo-maximum: Find all pixels with a higher value than a certain number of neighbor pixels.

3. Application of a mask to the pixels found in the previous step to extract the IR LEDs. The used mask is

\[
mask = \begin{bmatrix}
-2 & -2 & -2 \\
-2 & 15 & -2 \\
-2 & -2 & -2 \\
\end{bmatrix}
\]  

(4.1)

4. Find all pixels from the previous step that are over a certain threshold. These pixels correspond to IR LEDs.

It showed that this process still works although the resolution of the camera has changed. This is mainly compensated due to the new optic. The size of the robots in pixels on the image is almost the same as before with the old optic and lower resolution and therefore, the principle of the IR detection still works as before.

In comparison with the last version, the IR LED as well as the color LED is needed to detect the robots. Because of the construction of the robot, the IR LED and the color LED are attached at different heights in the robot. This causes problems in the edges where the relative position of the IR LED and the color LED is shifted because of the perspective of the camera. Therefore, both have to be projected to one predefined robot level which is chosen to be the height of the color LED. This means that the detected IR LEDs have to be projected to the color LED level which is in this case 17 mm below the IR LED level. This is done with simple geometric computation knowing the relative position of the color LED and the IR LED in respect to the camera by an intersection of the direction to the IR LED with the color LED plane. With this, it is possible to determine the correct pose of the robot independent of its position and orientation on the display area.

The effect can be seen on Figure 4.4 where the IR LED seems to lie inside of the left robot because of the perspective of the camera. For the right robot, whose orientation is in the other direction, the IR LED seems to lie outside the robot. The robot is defined as the illuminated circle created from the color LED. The robots captured in Figure 4.4 are at one corner of the display area where this effect is biggest.

As a result of the IR detection algorithm, a list with the positions of all detected IR LEDs is saved for further processing.
4.1.3 Robot extraction

To determine the position of the robots, different algorithms are applied and compared. The main requirement on these algorithms is a high accuracy at a low computation time. The accuracy is needed to control and track the robot precisely. The computation time is limited because of the requirement of real-time control. In the following, the tested algorithms are described.

Hough circle detection

The ‘Circular Hough transform’ is a method to extract circular shapes from an image. It is a modified version of the ‘Hough transform’ which is a technique to extract features from an image. It uses a voting procedure in the parameter space.

The Hough detection was initially used to detect lines in images. For that, two parameters, the distance to the origin and the angle of the line, are used. To detect circles, a modified version can be applied using three parameters: the center and the radius. Further information can be found in [12], [13] and [14].

The Circular Hough transform is out of the tried methods the most general way to extract circles from the image. The main disadvantage is that the algorithm is slow and for certain images (e.g., low exposure), the accuracy of the detections is not high enough.

Gradient image detection

The main idea behind the ‘Gradient image detection’ is to detect robots in the gradient image instead of the original image from the camera. The advantage if the gradient image is that borders (shape of the robot) can be found easier than in the original image because of its higher contrast. Especially robots with the color LED off can be detected easier in the gradient image. An example of such an image can be seen in Figure 4.5. For the detection of the robot, the list of detected IR LEDs is used. It is assumed that around each IR LED must be a robot. The following steps are applied for each IR LED that was found before:

1. Create a list of possible robot centers around the IR LED with a distance of the radius of the robot, which is known previously.
2. Take the median value of a certain number of pixels with a distance equal to the radius of the robot around each possible robot centerpoint and save that value for each centerpoint. It shows that the sum or the mean value of these pixels around the centerpoint result in worse results than the median value. This is because of the median value is not susceptible to outlier. For robots with the color LED on, the distance of evaluated pixels around the centerpoint was reduced to half of the radius to increase the accuracy.

3. The centerpoint with the highest saved value is assumed to be the most probable position of the robot. If another centerpoint with a high value is detected (for example if two robots are very close), this is also saved. The assignment and therefore the decision which detected robot corresponds to which IR LED is done in a subsequent step (see Section 4.1.4).

The main advantage of this method is that it works with robots that have the color LED on as well as with robots that have it off. The drawback is that the detection is not very accurate and in some situations, for example with light reflections, it fails completely. Another problem is that the computation cost to determine the gradient images is high.

**Illuminant search**

The 'Illuminant search' (Figure 4.6) uses the color LED to find the robot around each IR LED. The pixels at a distance equal to the radius of the robot are searched to find all values over a certain threshold. The robot center is then assumed to be in the center of consecutive pixels over the threshold value. If more than one possible robot is found, both are saved with weights (number of consecutive pixels) and assigned to the corresponding IR LED as used in the 'Gradient image detection'.

The main advantage of this method is that it is very fast. Furthermore, the detected robot poses are very precise and reflections do not disturb the detection. The disadvantage is that this method can only be used for robots with the color LED on. Because of its simplicity, stability and accuracy, this method is used in the final version.
4.1.4 Robot assignment

After finding possible positions of the robots and the IR LEDs, this information has to be matched so that each robot has one IR LED that indicates the orientation. In the Hough circle detection, for each detected robot, its corresponding IR LED is searched. The other methods use the opposite direction: starting from each IR LED, the corresponding robot is found. In general, there are two cases: One or two possible robots are detected that could correspond to each IR LED.

The assignment is done in an iterative way:

1. All IR LEDs where only one robot is detected around it are assigned to that one. Usually, this is the case for most of the robots.

2. All remaining IR LEDs (with two possible robot positions) are checked if still both corresponding robot positions are possible. This is done by checking if one position is already ‘occupied’ by an assigned robot. If that is the case, it is assumed that the other possible position is the correct one and it is assigned to that one.

3. Step 2 is repeated until a robot is assigned to every IR LED.

4. For the unlikely case that the robots form a circle and around each IR LED two possible robots are detected, the algorithm choses the most probable position for the first robot. After that, all other robots can be found with the iterative way described above.

The assignment process is show in Figure 4.7. The blue dots show the detected IR LEDs, the green dots indicate the most probable position of the robot (which is wrong in that case for the left robot), the yellow ones are the second most probable position (in that case only the left robot detected a second probable position). Around the right IR LED, only one possible position is detected where for the left one, both are possible. The assignment process is then assigning first the right robot (because it is unique). In the next step, one of the two possible positions of the left robot is already occupied. Therefore, it is assigned to the other one. In the very unlikely case of a closed circle of robots where all of them detected themself and the next neighbor as possible positions, the most probable position of the first robot is assumed to be the correct one. Usually, this shows to be the correct assumption. After that, all the other robots can be assigned as described above. For the case that the assumption was wrong, the robots get lost and have to be recovered as described in Section 4.3.
4.2 Tracking of the robots

As mentioned before, all robots look exactly the same. Therefore, it is impossible to differentiate them during the run. That means that the robots have to be detected at a previous initialization process and afterwards, they have to be tracked to know in every situation the exact position of each robot. If they get lost, for example due to human disturbance, they have to be recovered (see Section 4.3).

4.2.1 Initialization

In a first step, before the demo starts running, all the robots have to be initialized to know their start position. This is done by turning on all three IR LEDs of every robot one by one and get out of that its exact position and orientation. This process can be executed with maximum 7.5 Hz, which is half of the maximum frame rate of the camera. To reach this, the robots that are used have to be known before and every signal sent to the robots has to be received correctly and suddenly processed by the robot. In that case, it means that for expected 75 robots it takes 10 seconds to initialize all of them. In reality, it shows that this time is usually reached if all robots are on the display area. The theoretically highest detection rate of 15 Hz is not reached because due to the fact that the image is taken before a robot turns on the IR LEDs. So for each robot two images are taken. If an already detected robot does not turn off its IR LEDs before the next image is grabbed, the algorithm does not consider it and searches for other possible robot positions in the image. It can take up to 250 ms until the robot turns on the IR LEDs in the worst case. Because of that, the algorithm tries to detect the robot for 250 ms. Therefore, if the algorithm searches for robots that are not on the display area, it slows down the detection process. This means that the initialization time can be reduced if it is known before that some robots are not used so that the algorithm does not have to check them. If one robot is not found, it is assumed that it is not in the demo area and that it does not take part of the demo. Detected robots turn on their green LED to show that the detection worked properly.

After these steps, the robots should not be moved until the demo starts. Otherwise, they have to be detected again by the recovery process (see Section 4.3) during the demo.
Chapter 4. Tracking

4.2.2 Tracking process

The tracking process is mainly divided in three parts:

1. Robot detection to get the poses of all robots but without knowing which one is which
2. Pose estimation knowing their old pose and velocity commands
3. Matching of detected and estimated poses

The goal is to do the tracking as simple as possible to save computation power.

Robot detection

The robot detection is done as described in Section 4.1. After that, the detected positions of all robots are known but it is not yet known which detection corresponds to which robot.

Pose estimation

The pose estimation uses the knowledge about the position of the previous time step (resp. the initial position for the first time step) of each robot. Additionally, the velocity that has been sent to the robot in the last time step is known. With this information, it is possible to make an estimation of the new robot pose. This estimation does not represent the exact actual position of the robot due to noise in the measurement and in the model of the robot (see Section 3) as well as dynamic effects like accelerations.

Matching

The matching process fuses the information from the detection and the estimation. This could be done in the same way as the goal assignment. It shows that it can be done more efficient and simple because the detected and the estimated pose are usually very close. Due to simplicity, the closest pose estimation and detected pose are matched in each case if the distance is below a certain threshold. This could result in many mismachings but due to the fact that the movement during one time step is smaller than the radius of the robot, mismatching does not happen during normal demos. In Figure 4.8, the principle can be seen but for larger movements than in one time step can be reached.

Experiments show, that the distance between the estimated and the detected pose is usually smaller than 5 mm, which is not much worse than the accuracy that is possible with the used setup (see Section 4.1.1). The error is larger during acceleration or breaking phases because the pose prediction does not consider dynamic effects. This results in an error which can be up to 2 cm for the very short acceleration time. But this is still smaller than the radius of the robot and does not affect the reliable tracking.

If the robots are moved manually, it can happen that two robots are detected in the same position. In that case, both are recovered one after the other (see Section 4.3).

It shows that this simple matching is very fast and the accuracy is sufficient in this case. The distances that the robots drive within one time step are short enough to assure correct matching. Therefore, the matching process works and more complex tracking algorithm are not needed.
4.3 Recovery

If no robot is detected within the expected threshold of the estimated position, it is assumed that this robot is lost and it has to be recovered. In a first step, all lost robots are stopped. Due to the fact that all robots look the same, a certain distinctive feature has to be used to find them. It shows that the best and easiest way is to turn all IR LEDs of the lost robot on which results in a triangle of known shape. This can be seen in Figure 4.9. After that, this triangle is searched within the LEDs and if it is detected properly, the robot is recovered and it continues its task. This procedure takes normally 2 time steps, which is in this case 0.2 seconds. If the robot is not detected after 5 time steps, the algorithm chooses another lost robot and tries to recover that one.

If more than one robot is lost, they have to be recovered one by one. This is done by immediately stopping all robots that are lost and put them on a waiting list to be recovered. This waiting list is then processed until all lost robots are recovered. The order of recovery is organized in the following way: Robots that have never been recovered since they are lost have first priority. In a second step, robots that have already been in a recovery process but were not found are processed in a random order. This order avoids that potentially completely removed robots do not block the recovery process for newly lost robots. The check if new robots are lost is done in every time step, regardless of whether another robot is in the recovery process or not.

The whole recovery algorithm is only executed if at least one robot is lost respectively at least one is on the waiting list to be recovered.
Chapter 5

Autonomous Recharging

Charging is one of the main issues that have to be considered when using a large number of robots. Charging them manually would take a lot of time and the demo could not run for a longer time. Therefore, an automatic charging system is used. Around the display area, several charging slots are installed where the robots can charge themselves.

5.1 Charging station

A charging station that is used to charge the battery of the robots can be seen in Figure 5.1. Each charging station is comprised of three charging slots and can therefore charge three robots at once. A charging slot has two uprising wires mounted on both sides of a socket in the center (see Figure 5.2 left). The socket guides the robot at the very last approaching phase into the slot. On the bottom of the robot, there are two gold-plated contact areas (see Figure 5.2 right). If the robot has arrived completely in the charging slot, the uprising wires of the charging stations have contact to the gold plated contact areas of the robot and the charging process starts. To assure good contact, the chargers are mounted on a metallic sheet which pulls the very light weighted robots down.

It shows that the robot misses the charging slot sometimes at the first try and has to retry it. This is mainly because of detection errors, the limited resolution of the camera, the controllability of the robot and its nonholonomic constraints. An attempt to improve the rate of successful approaches is to install an additional guidance that guide the robot into the charging slot if it does not approaches very precisely. This guidance can be seen in Figure 5.3. This improved the success rate for older versions of the controller but for the newest version, the improvement is very small. A detailed comparison can be found in Section 6.1.6.

For practical reasons, the charging stations are positioned around the display area and therefore also at the edges of the camera image. There, the distortion of the image is higher and the resolution is a bit lower, which makes it more difficult to guide the robots precisely.
Chapter 5. Autonomous Recharging

Figure 5.1: A charging station with three charging slots

Figure 5.2: Left: A charging slot with two uprising wires for the contact to the robot on both side of the socket. Right: The bottom of the robot with gold plated areas for the contact with the wires of the charging station.

Figure 5.3: Additional guidance for the charging stations to guide the robots in the charging slots
5.2 Recharging algorithm

The main controller receives in every time step a status signal from each robot. The value of this signal corresponds to the remaining voltage in the battery. With this information, the controller assigns the robots with low battery to available charging stations. This is a process with several steps (see also Figure 5.4):

1. **Start:** The robot leaves the actual pattern. From now on, the robot still uses collision avoidance but is not considered in the goal assignment anymore. The remaining robots rearrange so that no ‘holes’ arise in the pattern.

2. **Approaching phase 1:** The robot follows predefined waypoints to a position in front of its charging slot. This position has a safety distance to the charging station so that it is prevented that the robot approaches the station from the opposite direction.

3. **Approaching phase 2:** If the distant position is reached, the robot approaches its charging slot up to a position right in front of the slot. After that step, the collision avoidance is disabled. This is required because the charging slots are very close and the robot should not be affected by other ones that are charging at the neighbor slots.

4. **Rotation:** Each goal position is just defined by the coordinates but not by the orientation. This means that the robot has to be rotated to its correct orientation. This is done by rotation the robot on the spot until the correct orientation is reached.

5. **Final approaching phase:** The goal position is continuously shifted towards the charging slot. The robot follows this goal. During that, the orientation is observed all the time to avoid that it rotates too much. If the rotation exceeds a threshold, the movement is interrupted and the robot rotates back to reach the correct orientation.

6. **Retry:** If after a certain time the charging slot is not reached, the robot restarts the process. This can happen if it gets stuck or if it misses the charging slot.

7. **Charger reached:** If the charging slot is reached successfully and the charging starts. The charging station is assumed to be reached if the position and the orientation of the robot is within a certain threshold. Further, to increase the precision, the position of the IR LED is also checked. To assure a stable contact, the robot pushes for a short time against the charging station before it stops and turns off its LED.

8. **Check:** As soon as the charging slot is reached, the actual voltage of the battery is saved. After 30 seconds the voltage is measured again and compared to the voltage before. If there is no increase, it is assumed that there is a problem and the charging does not work. This can happen if the robot is not completely in the charging slot or if there is a contact problem with the uprising wire of the charging station. In that case, the robot leaves the charging station and starts another approaching process to a charging slot.

During the charging, the robot is still communicating with the system and sends the value indicating its battery level. If it is detected that the battery is full, the charging process is finished and the robot returns to the display area. There, the remaining robots rearrange to clear a place for the returning robot.
Chapter 5. Autonomous Recharging

Approaching phase:
- Approach to point in front of charger
- Avoid obstacles

Rotation:
- $\vec{v} = 0$
- $\vec{\omega} = f(\text{angle})$
- NH-ORCA deactivated

Final approaching phase:
- $\vec{v}$: limited speed
- $\vec{\omega}$: limited angular speed
- check orientation continuously
- NH-ORCA deactivated

Charger reached, if:
- Robot position within threshold
- Robot orientation within threshold
- IR LED withining threshold
  $\Rightarrow$ Turn off and charge

Figure 5.4: Steps showing a robot approaching a charging station.
As mentioned, the controller decides which robot is charged, when it is charged and for how long the charging lasts. This decision depends on the mode that is desired. The display can be used in the following different ways:

**No charging**

If the full number of robots is needed, the charging can be deactivated completely. In this case, a robot with low battery just stops at some point and it cannot reach a charging slot afterwards.

**Constant number of robots**

The number of robots in the display remains constant all the time. This means that a predefined number of robots is in the charging stations. If a robot is fully charged and the battery level of another one on the display is below a certain level, they are simply exchanged. With that, it is possible to display always exactly the same pattern and it is therefore assured that there are always as many robots as needed to display a certain pattern nicely.

**Maximum number of robots**

The number of robots is dynamically adjusted in this mode. Every robot is running in the display as long as the battery is above a certain level. If the voltage falls below that level, the robot drives to a free charging slot and starts charging. As soon as the battery is full again, it goes back to the display. With that, the number of robot changes continuously. This leads to pattern with as many robots as possible but the goal positions have to be known for all the possible number of robots and they have to be changed whenever a robot leaves the display or if one comes back.

**Charge all robots**

In this mode, all robots are charged. The robots that are not in the charging station just wait at a waiting position. If all robots are charged fully, the program stops. This can be useful if for a specific pattern, all robots are needed.

The decision when the controller allows the robots to exchange, that means to leave the display or to come back from the charger, is another issue. It mainly depends on the program that is running. Generally, it has to be avoided that leaving robots disturb the image or animation that is displayed at that moment. For that, robots are usually exchanged before or after animations or between different patterns that are displayed. Further, not just the battery level is considered but also the distance to the next free charger. If this distance is too large and the robot still have some energy left, it continues driving until it comes closer to a free charging station or the battery level further decreases below a critical value.
Chapter 6

Results

6.1 Horizontal

The majority of the tests are done on a 2 x 2 m horizontal surface. At the edges of the surface, charging stations are installed which lowers the usable space for the display demo to about 1.8 x 2 m. In the following sections, different test scenarios are described. The goal positions for them are generated with a Voronoi algorithm that was implemented in the previous work [1] and [2] in Matlab. The goal positions are generated for all possible number of robots so that the pattern is still visually appealing (without gaps) if some robots are charging or do not take part of the demo at all. The parameters that are used can be seen in Table 6.1.

6.1.1 Trajectory on a square

In a first test, just one robot is used at the same time. The robot drives on the edges of a square given by its corner positions. Each side is recorded individually by setting the robot to the correct orientation in direction to the next goal. This is done to avoid that the controller automatically tries to make smooth rotations at the corners. The saved trajectories of the three tested robots are plotted in Figure 6.1. First, it can be seen that each robot has a different behavior. Small asymmetries in the robot or dust make the robot driving on a trajectory that differs from the direct straight line to the goal. The controller is programmed to guide the robot on a smooth and direct trajectory to the goal. Therefore, the robot does not try to get back on the direct path between starting position and goal, it tries to directly reach the goal.

In comparison with the same test on a vertical surface (see Section 6.2.1) where the gravitation is not in the same direction on every side of the square, there is no difference between the four sides on the horizontal plane. The average deviation of the trajectories to the direct straight line is 5 mm, the maximum deviation is 170 mm.
### Chapter 6. Results

Table 6.1: Parameters used for horizontal usage

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Value</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$v_{\text{max}}$</td>
<td>0.38</td>
<td>m</td>
<td>Maximum velocity</td>
</tr>
<tr>
<td>$v_{\text{pref}}$</td>
<td>0.2</td>
<td>m/s</td>
<td>Preferred velocity</td>
</tr>
<tr>
<td>$k$</td>
<td>8.0</td>
<td>1/m</td>
<td>Approach vel. param. $v = v_{\text{pref}} \times \min(1, k \times \text{dist}_\text{to goal})$</td>
</tr>
<tr>
<td>$T_{\text{orient}}$</td>
<td>0.2</td>
<td>s</td>
<td>Time to achieve correct orientation</td>
</tr>
<tr>
<td>$dx_{\text{at goal}}$</td>
<td>0.01</td>
<td>m</td>
<td>Threshold at goal</td>
</tr>
</tbody>
</table>

![Driving on a square: Horizontal](image)

Figure 6.1: Recorded trajectories of three different robots driving clockwisely on a square (black) given by its corner points on horizontal surface. The trajectories are recorded individually.
6.1.2 Collision avoidance test

In order to test the collision avoidance algorithm at an extreme situation, the goal assignment algorithm is not used for these first tests. The robots start in a circle (or in up to three concentric circles for a large number of robots) and try to reach the opposite position of the circle. Due to the local collision avoidance, this results in congestion in the center. This congestion gets worse, the more robots are used. The tests showed that with 75 robots, it takes usually 1.5 – 2 minutes until the congestion is dissolved and each robot reaches the opposite side of the circle. But it is shown that even with many robots, the collision avoidance is able to dissolve the congestion.

Figure 6.2 shows an example with 75 robots. The color of each robot is in this example dependent on the distance to the center. Starting from green at the exterior of the circle, it gets red to the center. Other transitions like a color transition from left to right or a color wheel are also possible. It can be seen that after the start, all robots try to drive towards the center point of the circle to reach their goal position on the opposite side on the shortest way. It shows the importance of having the goal assignment algorithm to avoid congestions.
Figure 6.2: Collision avoidance. The running time can be seen below every image. The color of the robot is a function of the distance to the centerpoint of the circle.
6.1.3 Static pattern formation

To see the performance of the algorithms and methods used for this project, several tests are done. For that, the robots display different patterns. For all patterns, all 75 robots were used to display the image.

Figure 6.3 shows the formation of a simple circle from a random initial position of the robots. The images are in chronological order, from left to right and from top to bottom. The total time to form this pattern is about 4 – 6 seconds, depending on the staring positions.

![Figure 6.3: Formation of a circle with 75 robots. The running time can be seen below every image.](image)

The goal assignment algorithm presented in Section 2.1 assures that each robot is assigned to that position that the global costs are minimal. This also allows taking any robot out of the pattern and putting it to any place on the display area. After it is recovered (see Section 4.3), all the robots are automatically reassigned and the repositioned robot can quickly reenter the formation at a close edge of the pattern. The other robots automatically move up so that the pattern is complete again. This can be seen in Figure 6.4 where nine robots are moved manually to the other side of the circle. Then, they are recovered and rejoin the pattern where the other robots move up.

Figure 6.5 shows a pattern displaying a worm in an apple. Left, the original image is shown. The right image shows the positions of the robots after running the Matlab simulation. The image on the bottom shows the real robots forming the pattern. It
Figure 6.4: Goal assignment: Nine robots are moved manually to the other side of the circle, are recovered and rejoin the pattern where the other robots move up. The running time can be seen below every image.
Figure 6.5: Image of a worm in an apple. The time to create the picture with the 75 real robots starting from a random position is about 5 seconds.

can be seen that the positions of the robots in the simulation correspond with the positions of the robots on the real display.

In Figure 6.6, several simple patterns are shown, in Figure 6.8, the formation of different images can be seen. Each of them takes about 4 – 9 seconds to form.

Figure 6.7 shows the trajectories between different static patterns. It can be seen, that the trajectories are not straight lines in every case because of noise or because a robot has to avoid another one.
Chapter 6. Results

(a) Square
(b) Circle
(c) Circle and bar
(d) Star

Figure 6.6: Robots displaying different simple patterns

−0.5 0 0.5
−0.8 −0.6 −0.4 −0.2 0 0.2 0.4 0.6 0.8
Trajectories
x [m] y [m]

(a) Square to Circle
(b) Circle to Triangle

Figure 6.7: Trajectories between static patterns with 75 robots. Red: starting positions, blue: goal positions
Figure 6.8: Robots displaying different patterns from images. Left: original image or Matlab simulation, Right: robot display. It can be seen that it is hard to recognize complex patterns like the Mouse or the Dog. Further, it can be seen that there are too many robots displaying the dog (overlapping of the goal positions). The color of the images are changed because it is not possible to display black.
6.1.4 Dynamic pattern formation

For the Dynamic pattern formation, the goals are moved in every time step. This makes it possible to move and transform every pattern presented above. In Figure 6.9 a circle formed of 75 robots moves at a constant velocity. At the same time, it is rotating and linearly expanding.

It is also possible to manually move robots. As in the static pattern formation, they rejoin the pattern after they are recovered.

Figure 6.9: Dynamic pattern formation: Circle with 75 robots that is shifted linearly, rotated and expanded. Presented in chronological order, from left to right and from top to bottom. The time between the images is always 2.5 seconds.
6.1.5 Video display

An extension of the previous tests is the video display. To realize a video display, an animation of consecutive pictures is given. From these, the goal positions for each frame are precomputed. During the display, the frames are processed successively which results in a smooth movement. The pictures of the video are the same as used in [1]. The animation consists of three parts. First, a waking human is displayed, then a mountain and at the end a bee flying to a flower. The first frame of each part can be seen in Figure 6.10. Figure 6.11 and Figure 6.12 show captures from the robots displaying the video. In this section, the experiments are done with 50 robots.

To achieve a visually appealing video, it has to be run quite slowly at the moment. If it runs faster, there are several problems occurring. First, due to the inaccuracy of the robots, they do not drive in direct straight lines to their goal positions. Therefore, if a goal position moves fast, it can happen that this robot is always quite far away from its goal position. If this happens with several robots, the visual impression is bad. Second, the goal assignment can cause effects that do not look good in the video. If, for example, the bee in Figure 6.12 turns fast, the goals are reassigned to minimize the total cost. Even though this is correct, it looks strange because the bee does not turn but transforms itself that the head becomes a wing, the wing part of the body and so on. To avoid this reliably, the algorithm would have to be changed. Some improvements are possible if the hysteresis value of the auction algorithm is lowered so that the goal assignment is just changed if there are larger changes in the image.

Figure 6.10: First frame of each part of the video [1]. Robots displaying this video can be seen in Figure 6.11 and Figure 6.12
Figure 6.11: Video display with 50 agents (1/2) presented in chronological order, from left to right and from top to bottom. The time between each image is about 2 seconds. The first frame (original image) of each part can be seen in Figure 6.10.
Figure 6.12: Video display with 50 agents (2/2) presented in chronological order, from left to right and from top to bottom. The time between each image is about 2 seconds. The first frame (original image) of each part can be seen in Figure 6.10.
6.1.6 Charging

As mentioned in Section 5, the approach of the robot to the charging station does not succeed every time at the first trial. Figure 6.13 shows a statistic of how many trials were needed until the robot successfully reached its charging station with and without additional guidance. For this experiment, different robots were used. It can be seen that most of the times, the robot succeeds the first time. The maximum number of trials in this test was four but it could be theoretically higher. For that reason, the robots leave the display before their battery is almost zero to assure that they still have enough power to reach the charging station even if they need more than one trial.

It can be seen that the additional guidance does not improve the result significantly. This is mainly due to the improved velocity controller of the robot and the improvements in the accuracy of the detection algorithm.

![Approaching the charging station](image)

Figure 6.13: Statistics about the number of trials needed to successfully reach the charging station with and without additional guidance
Table 6.2: Parameters used for vertical usage

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Value</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$v_{max}$</td>
<td>0.38</td>
<td>m</td>
<td>Maximum velocity</td>
</tr>
<tr>
<td>$v_{pref}$</td>
<td>0.2</td>
<td>m/s</td>
<td>Preferred velocity</td>
</tr>
<tr>
<td>$k$</td>
<td>30.0</td>
<td>1/m</td>
<td>Approach vel. param. $v = v_{pref} \times min(1, k \times dist_{to goal})$</td>
</tr>
<tr>
<td>$T_{orient}$</td>
<td>0.2</td>
<td>s</td>
<td>Time to achieve correct orientation</td>
</tr>
<tr>
<td>$dx_{at \text{goal}}$</td>
<td>0.03</td>
<td>m</td>
<td>Threshold at goal</td>
</tr>
</tbody>
</table>

6.2 Vertical

For the vertical usage, not as many tests as in the horizontal surface have been done, mainly due to lack of time. The vertical surface is made of a 0.75 mm metallic sheet glued on a wooden plate. The metallic sheet with a total width of 1.5 meters and a height of 2.3 meters is needed for the magnetic wheels of the robots. Charging stations are attached at the bottom of the area. There is also the option to put additional charging stations to the lower sides and to the top of the vertical display area.

The detection of the robots on the metallic sheet is more difficult than on the wooden plane on the floor because of reflections. With a low exposure of the camera, most of the reflections are not visible. Only lamps that are very close and directly illuminate the surface have to be turned off or covered in the direction to the surface because they are much brighter than the LEDs of the robots.

The main challenge of the extension to the vertical surface is the gravitational force on the robots. To compensate this force, each robot has an internal accelerometer that allows detecting the orientation of the robot on a vertical wall and this allows therefore to the controller to compensate the gravitational force. Further, some parameters have to be adjusted, for example the approach velocity to the goal is increased so that the robot has enough power to reach the goal even if it approaches it from below. The parameters that are used can be seen in Table 6.2.

As in the horizontal surface, different tests and experiments are done. The software to control the swarm is the same as in the horizontal surface and therefore, the same displays are possible. Below, some of the tests are described.

6.2.1 Trajectory on a square

As on the horizontal surface, the first test is done just with one robot at the same time. The goals of the tests are the same as in the horizontal experiments: The robot drives on the edges of a square given by its corner positions and each side is recorded individually by setting the robot to the correct orientation in direction to the next goal. The saved trajectories of the three robots are plotted in Figure 6.14. For this test, the same robots as for the identical test on the horizontal surface are used. It can be seen that the trajectories are worse than in the horizontal tests. Further, the differences in the performance of the robots are larger on the vertical surface. It is also possible to see differences between the four sides of the square.

Table 6.3 shows the maximum and the average deviation of the robots from the direct straight line for this experiment. It can be seen that downwards driving robots (right side of the square in Figure 6.14) are much closer to the straight line than the upwards driving (left side of the square in Figure 6.14) or the sideways driving ones.
Driving on a square: Vertical

Figure 6.14: Recorded trajectories of the three robots that were already used for the horizontal test (see Section 6.1.1) driving clockwisely on a square (black) given by its corner points on vertical surface. The trajectories are recorded individually.

Further experiments also show that there are large differences between the robots. Choosing just the 'best' robots would lead to much better results.

Table 6.3: Deviation of the robots to the direct straight line on the vertical surface depending on the driving direction

<table>
<thead>
<tr>
<th>Driving direction</th>
<th>Maximum deviation</th>
<th>Average deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Downwards</td>
<td>255 mm</td>
<td>47 mm</td>
</tr>
<tr>
<td>Upwards</td>
<td>310 mm</td>
<td>78 mm</td>
</tr>
<tr>
<td>Sideways</td>
<td>320 mm</td>
<td>61 mm</td>
</tr>
</tbody>
</table>

6.2.2 Collision avoidance test

The collision avoidance test is done in the same way like on the horizontal plane but with maximum 50 robots. The full amount of robots was not tested because of the limited space on the vertical surface, the lower accuracy of the control and the oscillating behavior in the current version. This would result in collisions with many robots. Further, some robots had technical problems and could not be used for some tests. Even though the algorithms still work on the vertical surface, the collisions become more frequent. In these tests, another problem showed up: some robots start running even if they do not receive a signal just because of the gravitational force. Where for some robots the friction in the non-moving wheels is high enough to hold them, others start moving downwards and, as a result of that, cause collisions.
6.2.3 Static pattern formation

The static pattern formation on the vertical wall was tested with up to 75 robots. It shows that if all robots are used, it takes long until every single robot reaches its goal position because of the oscillation of the robots and because the display area is very crowded. For that reason, the number of robots was lowered for some more tests. In general, the static pattern formation also works on the vertical plane. As in the other vertical tests before, the robots need more time to finally reach their goal positions and the movement is more noisy and shaky.

Figure 6.15 shows some static patterns displayed on the vertical surface. It can be seen that, compared with the horizontal surface, the patterns and objects are displayed worse. This is due to the higher threshold at the goal position (to assure that the goals are reached) and also because of the lower number of robots.

Figure 6.16 shows the trajectories between different static patterns on the vertical surface. It can be seen that the trajectories are much more noisy than on the horizontal plane.

Figure 6.15: Static pattern formation on the vertical surface displaying different patterns and objects.
6.2.4 Dynamic pattern formation

As for the horizontal display, in the Dynamic pattern formation, the goals are moved in every time step. This makes it possible to move and transform every pattern presented above. In Figure 6.17 a circle formed of 14 robots moves at a constant velocity. At the same time, it is rotating and linearly expanding. It can be seen that the shape of the circle is less accurate than on the horizontal surface.

It is also possible to manually move robots. As in the static pattern formation, they rejoin the pattern after they are recovered.

Figure 6.16: Trajectories between static patterns on the vertical plane with 20 robots. Red: starting positions, blue: goal positions
Figure 6.17: Dynamic pattern formation on the vertical surface displaying a circle with 14 robots.
6.2.5 Video display

Due to the smaller space on the vertical surface and the lower accuracy, the video display is just tested with 14 robots. It shows that by taking the robots that have the best behavior on the vertical surface, it is possible to display the complex video on the vertical surface. Compared to the vertical surface, the movements are not very smooth and the objects in the video can get distorted. Nevertheless, the animation is visible. To get a better video display on vertical surfaces, the movements of the robots have to be improved that they are smoother.

Figure 6.18 shows four frames of the first part of the video (walking human) presented in Section 6.1.5. It can be seen that the robots do not follow the goal positions as good as in the horizontal surface. This distorts the image.

![Vertical video display with 14 agents presented in chronological order, from left to right and from top to bottom. The time between each image is about 2 seconds. Here, just the first part of the video with the walking human is shown.](image)
6.2.6 Charging

Tests with autonomous charging on the vertical surface show that if the chargers on the bottom are used, the success rate is almost as high as in the horizontal surface (Section 6.1.6). The contact between the charging station and the robot is even better than on the horizontal plane. The magnetic force and the gravitation pull the robots to the charging station. For the additional possible chargers on the sides and on top, no tests are done. It can be expected that there, it is much more difficult for the robots to reach the charging slot (driving downwards is more precise than upwards or sideways as mentioned in Section 6.2.1) and to have good contact to the station.
Chapter 7

Conclusion and Outlook

In this chapter, a conclusion of the real-time control of a large swarm of mini-robots is given. The main results and outputs of the project are concluded and an outlook of possible extensions and optimizations of the projects is presented.

7.1 Conclusion

In this project a novel robot display with a swarm of 75 small autonomous mobile robots is developed. Together, they are able to display images and videos where each robot represents a pixel of the image. Starting with a working version with 14 e-puck robots the algorithms are adapted, extended and optimized to achieve real-time control of the larger swarm. Compared with the previous version, the larger swarm makes it now possible to display much more detailed pictures and animations.

There are various challenges arising with the upscaling to a large swarm of low-cost mini-robots. The main work packages that are done for this project are the following ones:

- In a first step, the robots are characterized. It shows that the new robots are more difficult to control than the previous ones that used internal wheel encoders and were therefore able to navigate very precisely. The characteristics of the new robots are not exactly the same for all of them and they are not able to drive on a straight line in open loop control even if the internal velocity controller is used. Nevertheless, it is possible to compensate this behavior with the closed loop controller and navigate them mostly collision free.

- Second, the detection of the robots is adapted to the new robots. One major challenge is that the robots are not distinguishable anymore and they have to be tracked. It shows that the tracker works reliably even with a large number of robots. During normal run, robots usually just get lost if there is an external disturbance. Due to the recovery algorithm, lost robots can be recovered very quickly and the demo is not affected by that. Nevertheless, it fails if there are large changes in environment illumination.

- Third, large amounts of robots also mean new challenges with the general handling of them. Switching them on and off and charging them gets very time-consuming with a large swarm of robots. Therefore, the robots are complemented with a sleep mode that allows them to go in a state where it uses very few energy. This allows leaving the robots a longer time running and
they have not to be switched off manually every time. The charging problem is solved by charging stations where each robot can charge itself during the demo if its battery level is too low. This theoretically allows running the demo for a long time without interacting. In this work, an algorithm for this is developed.

• Fourth, the algorithms need much more computation power for a large number of robots. To handle that, a faster computer is set up for the demo, the most computational expensive parts of the code are located and the code was extended to multi-threading. The code is split up in three main threads, one for image processing, one for sending the signals and one for the rest. Further, the collision avoidance algorithm is divided in several threads to increase the computation speed. Another very computational expensive part is the goal assignment algorithm which is changed to a distributed auction algorithm that is much faster than the previous one.

• At the end, the demo was ported to use on a vertical surface. Even though the main algorithms are the same, it shows that there are many additional challenges arising for the vertical usage. This is mainly because of the advanced and less precise controllability of the robots on a vertical surface. Due to lack of time, it was not possible to make the demo run perfectly on the wall. Nevertheless, simple demos with a couple of robots are possible.

It is shown that it is possible to control a large swarm of cheap mini robots in real-time and display different patterns and videos.

7.2 Outlook

This project can be evolved much more. Some fields where improvements could be possible are presented here.

One of the main goals of this project was to make the displayed images better by using more robots. Even though the number of robots is much higher than before and normal patterns, symbols and images can be displayed in a nice way, it is still not possible to display very complex images and patterns. For that, much more robots are needed.

With even more robots, the required computation power increases much more. Instead of using an even faster computer, it could be possible to distribute the algorithms to the robots. This would lead in a new way of controlling the robots where each robot has more intelligence than in the current version.

It also showed that with the current version, the vertical display is very limited. To make that better, improvements of the controller and the robots are needed. To commercially use the Display Swarm, it is required that it not just works perfectly on a horizontal surface but also on a vertical wall.

To improve the visual effect of the swarm display, it would be possible to equip future robots not just with a color LED pointing upwards but also one that illuminates the floor under and around the robot. This would give a nice effect where not just the pixels represented with the robots are illuminated but also the area around the pixels. This would lead to more intense and richer colors and images.
One weak point of the current version is that just robots with a turned on color LED are detected reliably. Even though it is usually the case that the color LED is turned on, it would be nice if it also worked without the color LED, for example to leave the display for charging.

To use the project as a mobile and portable demo, it is needed to include a very fast calibration of the camera. This would allow to carry the demo with a limited number of robots to any place and quickly show it.

As mentioned in the introduction, the demo would be even more attractive if there was an interaction between a human and the robots. This would allow the people to play with the Display Swarm and therefore show the whole potential of this technology.

An even more challenging extension would be to bring the robot display to 3D. Then, impressive three dimensional objects could be displayed or also effects like flying birds in a large swarm could be shown.
Bibliography


Appendix A

Datasheet robots
Speed and power control for the Elisa robot

Speed control

In order to implement a speed control, a measure of the current velocity given by the motors is needed; for this purpose the back-emf is sampled. This measure is then compared to the desired velocity to obtain an error that will feed to the controller. The following schema illustrates the controller (closed-loop):

The controller used to control the speed is a PID controller plus a feed forward term (kFF), where:

\[ E = desired \text{ speed} - measured \text{ speed} \]

\[ \text{duty} = kFF \times desired \text{ speed} + P \times E_i + D \times (E_i - E_{i-1}) + I \times \left( \sum E \right) \]

The sum of the errors used with the I term is kept within a predefined range (cannot grow infinitely). After parameters tuning the coefficient are chosen to be: P=40.0, I=1.0, D=10.0, kFF=120.0.

The following two figures show the behavior of this controller for various speeds (till the maximum) in free-run mode and when the controller is stressed respectively.

From the first figure it's possible to notice that at high speed the controller isn't able to reach the desired speed, but this is not a problem since these velocities will probably never be used in the final application.

In the second figure at the beginning the motors run freely, then the left and right motors are forced to stop and then released again various times.
Vertical speed control

When the robot tries to move against the gravity force it will need more power to maintain the same speed, conversely when it moves downwards it must reduce the power given to the motors to maintain the velocity. This is accomplished by the speed controller adapting the feed forward term (kFF) based on the current angle. The parameter adaptation starts automatically when the robot is tilted more than about 45 degrees.

Power control

A general power limitation of 60% (100% given from the user means 60% on the robot side) is always applied to not stress the battery and heat the motor.

In order to implement a power consumption control, an estimation of the current power consumption given by the motors is needed; for this purpose the voltage drop on the motor driver is sampled. Once the current power estimation is obtained it's necessary to have a rule that specify a maximum power consumption for a certain velocity, independently of the terrain the robot is moving on. To get this rule the estimated power consumption when the motor is free-running and when the motor is forced to stop completely is taken for various velocities. The limit is then fixed to be the average of the two

\[
\frac{\text{max consumption} - \text{min consumption}}{2}
\]

The following chart shows the current measure of the motor with a pwm duty of 50% (50% of 60% means actually 30% of the motor period); at the beginning the motor is moving freely, then it was stopped.
It's worth to note that the power control is deactivated when the desired velocity is smaller than 20%; this way the robot should have the possibility to move in any situation. For instance if the desired speed is 30%, but the current consumption is too high then the speed is reduced, let's say to 20%; now can happen that the robot isn't able to move at this speed, so letting him consume a little more current than the one admitted normally, it will move. Another example is when the robot start moving, in this situation the power consumption is high, but the robot must be able to start moving anyway.

The limiter of the power consumption is a closed-loop controller as shown in the following schema:

The controller has the following properties:

- enabling conditions: velocity ≥ 20% and current power consumption for desired velocity > power consumption limit for that velocity

- \[ K = (\text{max consumption} - \text{current duty limit consumption}) \times \frac{3}{4} \]

- forward motion: \[ \text{duty new} = \text{duty desired} - K \]

- backward motion: \[ \text{duty new} = \text{duty desired} + K \]